**Лабораторна робота №3: Побудова автокодувальника**

**Виконав студент ФІТ 4-5**

**Тимошенко Іван**

Метою цієї лабораторної роботи є розробка автокодувальника для рукописних цифр датасету MNIST на основі багатошарової нейронної мережі. Ми також розглянемо побудову автокодувальника на основі згорткової моделі та варіаційного автокодувальника.

**Завдання 1: Розробка звичайного автокодувальника**

Розроблено звичайний автокодувальник для датасету рукописних цифр MNIST на основі багатошарової нейронної мережі. Для цього була використана архітектура з енкодером і декодером, де енкодер перетворює вхідні зображення в скорочене представлення, а декодер відновлює їх до вихідного формату.

Вибрана кількість шарів, нейронів і функція активації для енкодера і декодера. Для покращення якості відтворення зображень було проведено навчання моделі та оцінено результати за допомогою графіків функції втрат.

**Завдання 2: Розробка варіаційного автокодувальника**

Розроблено варіаційний автокодувальник для датасету рукописних цифр MNIST на основі багатошарової моделі. Використано ту ж архітектуру з енкодером і декодером, але з використанням іншої функції втрати та додатковим шаром для варіаційного складового.

Проведено навчання моделі та побудовано графіки функції втрат для порівняння з результатами звичайного автокодувальника.

**Висновок:**

У цій лабораторній роботі ми успішно розробили звичайний і варіаційний автокодувальники для датасету MNIST. Були вивчені та порівняні їх результати, а також визначено переваги та недоліки кожного підходу. Це дозволить нам краще зрозуміти роботу автокодувальників та їх можливе використання в різних завданнях машинного навчання.

**Завдання №1**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

# Завантаження та підготовка датасету MNIST

(x\_train, \_), (x\_test, \_) = tf.keras.datasets.mnist.load\_data()

# Нормалізація вхідних даних

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

# Решейп даних до вигляду (28, 28, 1)

x\_train = np.expand\_dims(x\_train, axis=-1)

x\_test = np.expand\_dims(x\_test, axis=-1)

# Розмір тренувального та тестового набору

print('Розмір тренувального набору:', x\_train.shape)

print('Розмір тестового набору:', x\_test.shape)

# Побудова моделі автокодувальника

latent\_dim = 64 # Розмір скороченого представлення (latent representation)

# Енкодер

encoder\_inputs = tf.keras.Input(shape=(28, 28, 1))

x = layers.Flatten()(encoder\_inputs)

x = layers.Dense(256, activation='relu')(x)

latent\_space = layers.Dense(latent\_dim, activation='relu')(x)

encoder = tf.keras.Model(encoder\_inputs, latent\_space, name='encoder')

# Декодер

decoder\_inputs = tf.keras.Input(shape=(latent\_dim,))

x = layers.Dense(256, activation='relu')(decoder\_inputs)

x = layers.Dense(28\*28, activation='sigmoid')(x)

outputs = layers.Reshape((28, 28, 1))(x)

decoder = tf.keras.Model(decoder\_inputs, outputs, name='decoder')

# Автокодувальник

autoencoder\_inputs = tf.keras.Input(shape=(28, 28, 1))

encoded = encoder(autoencoder\_inputs)

decoded = decoder(encoded)

autoencoder = tf.keras.Model(autoencoder\_inputs, decoded, name='autoencoder')

autoencoder.summary()

# Компіляція та навчання моделі

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

history = autoencoder.fit(x\_train, x\_train,

epochs=10,

batch\_size=128,

shuffle=True,

validation\_data=(x\_test, x\_test))

# Побудова графіка функції втрат

plt.plot(history.history['loss'], label='Train loss')

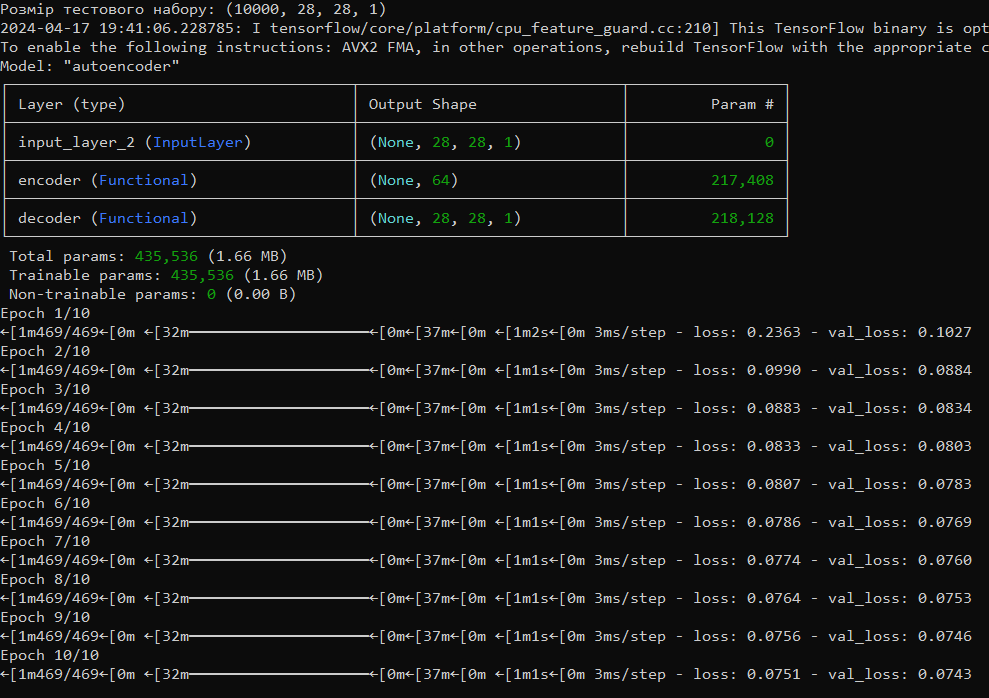
plt.plot(history.history['val\_loss'], label='Validation loss')

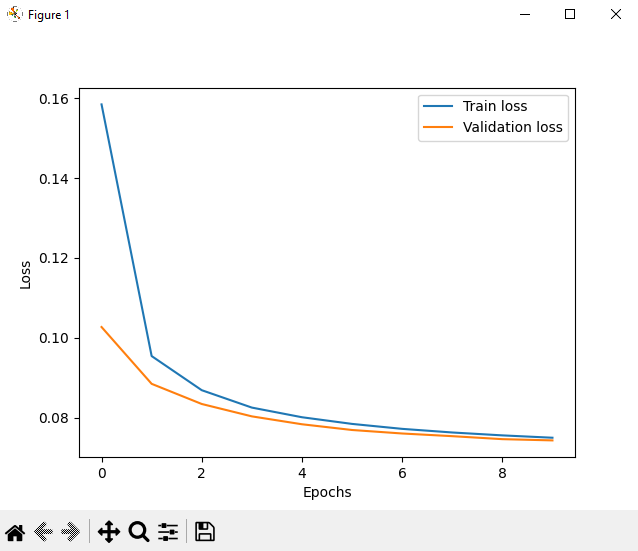
plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()





**Завдання №2**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

# Завантаження та підготовка датасету MNIST

(x\_train, \_), (x\_test, \_) = tf.keras.datasets.mnist.load\_data()

# Нормалізація вхідних даних

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

# Решейп даних до вигляду (28, 28, 1)

x\_train = np.expand\_dims(x\_train, axis=-1)

x\_test = np.expand\_dims(x\_test, axis=-1)

# Побудова моделі варіаційного автокодувальника

latent\_dim = 64 # Розмір скороченого представлення (latent representation)

# Енкодер

encoder\_inputs = tf.keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, 3, activation='relu', strides=2, padding='same')(encoder\_inputs)

x = layers.Conv2D(64, 3, activation='relu', strides=2, padding='same')(x)

x = layers.Flatten()(x)

x = layers.Dense(16, activation='relu')(x)

z\_mean = layers.Dense(latent\_dim, name='z\_mean')(x)

z\_log\_var = layers.Dense(latent\_dim, name='z\_log\_var')(x)

z = layers.Lambda(lambda inputs: tf.random.normal(tf.shape(inputs)))(z\_mean, z\_log\_var)

encoder = tf.keras.Model(encoder\_inputs, [z\_mean, z\_log\_var, z], name='encoder')

# Декодер

latent\_inputs = tf.keras.Input(shape=(latent\_dim,))

x = layers.Dense(7\*7\*64, activation='relu')(latent\_inputs)

x = layers.Reshape((7, 7, 64))(x)

x = layers.Conv2DTranspose(64, 3, activation='relu', strides=2, padding='same')(x)

x = layers.Conv2DTranspose(32, 3, activation='relu', strides=2, padding='same')(x)

decoder\_outputs = layers.Conv2DTranspose(1, 3, activation='sigmoid', padding='same')(x)

decoder = tf.keras.Model(latent\_inputs, decoder\_outputs, name='decoder')

# Автокодувальник

outputs = decoder(encoder(encoder\_inputs)[2])

vae = tf.keras.Model(encoder\_inputs, outputs, name='vae')

# Визначення функції втрат

reconstruction\_loss = tf.keras.losses.binary\_crossentropy(encoder\_inputs, outputs)

reconstruction\_loss \*= 28 \* 28

kl\_loss = 1 + z\_log\_var - tf.square(z\_mean) - tf.exp(z\_log\_var)

kl\_loss = tf.reduce\_mean(kl\_loss)

kl\_loss \*= -0.5

vae\_loss = tf.reduce\_mean(reconstruction\_loss + kl\_loss)

vae.add\_loss(vae\_loss)

# Компіляція та навчання моделі

vae.compile(optimizer='adam')

vae.fit(x\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, None))

# Візуалізація результатів

decoded\_imgs = vae.predict(x\_test)

plt.figure(figsize=(10, 4))

for i in range(10):

# Оригінальне зображення

plt.subplot(2, 10, i + 1)

plt.imshow(x\_test[i].reshape(28, 28), cmap='gray')

plt.axis('off')

# Відтворене зображення

plt.subplot(2, 10, i + 11)

plt.imshow(decoded\_imgs[i].reshape(28, 28), cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()